

國立交通大學

電機與控制工程學系

碩士論文

可應用各種強韌及高位元率之編碼方式的圖形式軟體浮水印架構

A Flexible Graph-based Software Watermarking Framework with Robust
and High Bit-rate Encodings



研究生：洪嘉良

指導教授：黃育綸 博士

中華民國九十五年九月

可應用各種強韌及高位元率之編碼方式的圖形式軟體浮水印架構

A Flexible Graph-based Software Watermarking Framework with Robust
and High Bit-rate Encodings

研究生：洪嘉良

Student : Chia-Liang Hung

指導教授：黃育綸 博士

Advisor : Dr. Yu-Lun Huang



Submitted to Degree of Electrical Engineering and Control Engineering
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of Master
in
Electrical and Control Engineering
September 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年九月

可應用各種強韌及高位元率之編碼方式的圖形式軟體 浮水印架構

研究生：洪嘉良

指導教授：黃育綸 博士

國立交通大學

電機與控制工程研究所



在這篇論文中，我們提出一套圖形式軟體浮水印架構，可以依據不同的安全需求，適度地採用適用的浮水印編碼演算法，以兼顧運作效能與安全度。現有的圖形式軟體浮水印演算法中，如 QP 及 QPS，僅能用於解決特定問題，因此無法提供一套全面性的解決方案，符合軟體浮水印在效能與安全方面的各種需求。此外，這些演算法具有低資料率與遭受惡意攻擊而破壞浮水印資訊等缺點。為了解決現有浮水印編碼演算法中的缺點，在我們所提出的軟體浮水印架構中，可以視執行效能與安全需求，利用提出的三種編碼演算法之一，將浮水印資訊嵌入於軟體模組中，以提供更高的資料率與強韌度。在這個浮水印的架構中，我們將會把程式轉換成圖形，並且將這個圖形分割成較為小的子圖，根據安全上或效能上的多樣需求，這些子圖將可以依照不同的編碼機制嵌入浮水印資訊。最後，我們分析並比較各種現存的軟體浮水印機制與本論文所提之方法，在錯誤偵測、抵擋惡性攻擊以及資料率等能力，以評估其強韌度與效能。

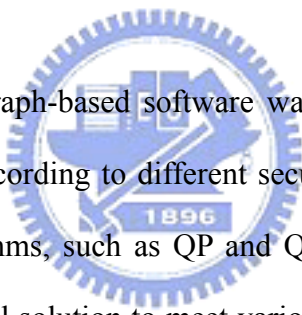
A Flexible Graph-based Software Watermarking Framework with Robust and High Bit-rate Encodings

Student: Chia-Liang Hung

Advisor: Dr. Yu-Lun Huang

Department of Electrical and Control Engineering
National Chiao-Tung University

Abstract

The logo of National Chiao-Tung University is a circular emblem with a blue border. Inside the circle, there is a stylized blue figure that appears to be a person or a symbol, and the year '1896' is written at the bottom of the inner circle.

In this paper, we propose a graph-based software watermarking framework to flexibly adapt hybrid encoding algorithms according to different security requirements. Existing graph-based software watermarking algorithms, such as QP and QPS, only address specific problems and thus cannot provide a one-fit-all solution to meet various requirements in terms of performance and security. In addition, these algorithms could suffer from the low data rate issue and vulnerable to additive and subtract attacks. To address the above issues, the proposed framework works in a hybrid manner and three encoding algorithms are also proposed to cooperate with our framework and to achieve higher data rate and robustness. In this paper, a software program is represented as a graph and can be further divided into smaller sub-graphs. The watermarking procedure runs through the graph and applies one of the three proposed encoding algorithms to each visited sub-graph per the security and efficiency requirement. As an evaluation of our work, error detection capability, attack resistance and encoding data rate are analyzed and compared between our work and the related work. The result shows that the proposed framework performs better bit rates under the same requirements.

謝 誌

在交大的日子，終於要劃下一個句號了，在這段短暫卻又快樂的時光，最感謝的莫過於黃育綸老師的栽培與提攜，不但提供了我們最好的研究環境，也總是在關鍵的時候，指點迷惘的我不管在研究上或是生活上的方向，也由於這些寶貴的意見，才有接下來的一字一句，雖然沒有達成老師百分之百的要求，但是盡力學習老師研究的精神，是我最大的收穫。此外，感謝實驗室裡同學及學弟妹們，總是提供我許多即時的助力，與你們一同砥礪琢磨，相互討論，使我獲益良多。感謝來自家人和朋友的鼓勵和體諒，是我最大的後盾。

兩年多來，我做到了一些，也有許多的不足，但是因為有你們，我會繼續的努力，因為有你們，才有烙印在我腦海中的美好時光。



Contents

摘 要	i
Abstract	ii
謝 誌	iii
Contents	iv
List of Figures	vi
List of Tables	viii
Chapter 1 Introduction	1
1.1 Background.....	1
1.2 Contribution.....	3
1.3 Synopsis.....	3
Chapter 2 Related Work	4
2.1 Graph Theoretic Approach for Software Watermarking.....	4
2.2 Static Watermark Algorithms.....	5
2.2.1 QP Algorithm.....	6
2.2.2 QPS Algorithm.....	7
2.2.3 Problems	9
2.3 Dynamic Watermark Algorithm	12
2.4 Summary.....	15
Chapter 3 The Proposed Software Watermarking Framework.....	16
3.1 Framework.....	16
3.2 Proposed Graph Encoding Algorithms	20
3.2.1 Link Encoding (LE).....	20
3.2.2 Color Encoding (CE)	23
3.2.3 Link with Color Encoding (LCE)	25
3.3 Example	28
3.4 Path Analysis	30
3.5 Error Detection	32
3.6 Summary.....	33
Chapter 4 Analysis.....	34
4.1 Security Analysis	34

4.2 Bit rate Analysis	36
4.3 Summary.....	37
Chapter 5 Comparison.....	38
5.1 Framework Characteristic.....	38
5.2 Bit rate of Software Watermark Algorithms	38
5.3 Characteristic of Software Watermark Algorithms.....	40
5.4 Summary.....	41
Chapter 6 Conclusion.....	42
Chapter 7 Future Work.....	43
Reference	44



List of Figures

Figure 2.1 Graph Theoretic Approach.....	5
Figure 2.2 An Example of QP Algorithm.....	6
Figure 2.3 QP Recognition Algorithm.....	7
Figure 2.4 Failure Recognition of QP Algorithm.....	8
Figure 2.5 Failure Recognition of QP Algorithm.....	8
Figure 2.6 Example of Recognition Problem in QP Algorithm.....	10
Figure 2.7 Example of Recognition Problem in QP Algorithm.....	11
Figure 2.8 Example of Recognition Problem in QP Algorithm.....	11
Figure 2.9 Example of Permutation Encoding Algorithm.....	13
Figure 2.10 Example of Radix Encoding Algorithm.....	14
Figure 2.11 Example of PP Tree Encoding Algorithm.....	14
Figure 3.1 Embedding Phase of Proposed Framework.....	17
Figure 3.2 Example of Path Analysis.....	18
Figure 3.3 Recognition Phase of Proposed Framework.....	19
Figure 3.4 Example of Proposed Embedding Phase Framework.....	20
Figure 3.5 Example of Link Encoding in Embedding Phase.....	21
Figure 3.6 Pseudo Code of LE Embedding Algorithm.....	21
Figure 3.7 Pseudo Code of LE Recognition Algorithm.....	22
Figure 3.8 Pseudo Code of CE Embedding Algorithm.....	23
Figure 3.9 Example of Color Encoding in Embedding Phase.....	24
Figure 3.10: Pseudo Code of CE Recognition Algorithm.....	25
Figure 3.11: Pseudo Code of LCE Embedding Algorithm.....	27
Figure 3.12: Example of Link with Color Encoding in Embedding Phase.....	27
Figure 3.13: Pseudo Code of LCE Recognition Algorithm.....	28
Figure 3.14: Example Program.....	28
Figure 3.15: The Parsed Program.....	29
Figure 3.16: The Graph of Embedding.....	29
Figure 3.17: The Watermarked Program.....	30
Figure 3.18: The Parsed Watermarked Program.....	30
Figure 3.19 Example of 4 Possible Paths with 4 Vertices and LE Algorithm.....	30
Figure 3.20 Example of 2 Possible Paths with 4 Vertices and LE Algorithm.....	31
Figure 3.21 Example of Zero Possible Paths with 4 Vertices and LE Algorithm.....	31
Figure 3.22 Example of Path Analysis with 4 Vertices and LE Algorithm.....	31
Figure 3.23 Example of Path Analysis with 2 Vertices and LE Algorithm.....	32
Figure 3.24 Example of Error Detection.....	32
Figure 4.1 Example of Vertex Subtractive Attack.....	35



List of Tables

Table 1 : Framework Characteristic.....	38
Table 2: Bit rate of Graph Encoding Algorithm	40
Table 3: Comparison of Graph Encoding Algorithm.....	40

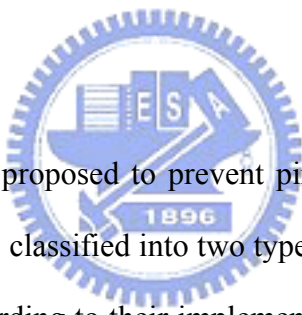


Chapter 1

Introduction

In the past few years, development of digital technology has enabled digital contents to be accessed over the Internet. Advance of modern network technologies makes electronic distribution of digital contents increasingly popular and meanwhile promotes the acceptance to the public. However, the facile distribution of digital contents also has side effects that make illicit copying and dissemination rather easier, for example, the controvertible mp3 download platform. In this chapter, we explain the background of software watermark.

1.1 Background



Recently, many methods were proposed to prevent piracy and prove the ownership of the digital contents. These methods can be classified into two types, software-based [9] [12] [13] [16] [19] and hardware-based [10] [17], according to their implementation. The software-based methods, used to prove the ownership, are implemented using pure software, for example, watermarking [20] [21] [23], fingerprinting [11] [18], birthmarking [6] [14] and so on. The hardware-based methods should be operated on a *trusted computing* platform which you can't tamper with the application software and where these applications can communicate securely with their authors and with each other. Digital Rights Management (DRM) is one of most famous schemes implemented on trusted computing platforms. Compared with software-based methods, hardware-based methods have better security but higher cost. In addition, hardware-based methods encounter a problem in deployment. In such a condition, software-based methods are widely used in protecting the digital contents.

There are two types of software watermarking algorithms, static and dynamic software watermarking algorithms, depend on the way they embed or recognize the watermarking

information. Static software watermarking algorithm is directly embedded and extracted the unique message. Dynamic software watermarking algorithm [15] uses functions in the program and the correct information will be embedded during the execution of the program. In the same way, the right information will also be extracted and identified.

A good software watermarking system must be evaluated using following criteria:

- **Robust:** Watermark with high robust can against various attacks as many as possible while maintains the integrity.
- **Bit rate:** The ratio of bits watermarked to the extra code size is called bit rate. The higher bit rate, the more bits can be embedded to the software module.
- **Stealth:** High stealth makes the piracy confuse with original and watermarked program.
- **Performance:** Watermarked program should maintain the same performance as the original.

In 1996, Davidson and Myhrvold [1] published the first software watermarking algorithm in order based. In this algorithm, watermark information is embedded by reorder the basic blocks of in the original program. In 1999, Qu and Potkonjak [2] proposed QP Algorithm, which is a graph-based software watermarking through register allocation. QP algorithm has three kinds of methods to embed the message: adding edges 、 selecting MIS (maximum independent set) and adding nodes. In QP algorithm, edges and vertices are added or connected in a graph according to the watermark information. Through the edges set and vertices set of the graph, the message can be extracted from the watermarked program. However, credibility in QP algorithm and security from attack hadn't been considered in their analysis.

Collberg and Thomborson [4] [7] [8] brought out the first dynamic software watermark algorithm, CT algorithm, in the same year of QP algorithm. CT algorithm implemented in Java called SandMark. Five kinds of graph encoding algorithms which have their respective features are applied in SandMark. Based on those graph encoding algorithms, CT algorithm have high robust and stealth against different kinds of attacks. In 2004, Myles and Collberg [3] implemented QPS

algorithm, an improved QP algorithm with SandMark. QPS algorithm rearranges the color of vertex when message is embedded to correct the problems in the embedding and recognition phases of QP algorithm. Color between vertices is used to detect and re-correct the embedded information in QPS algorithm. The characters of QPS algorithm, stealth and robust, are evaluated by QPS-based SandMark.

1.2 Contribution

In this paper, we not only improve QP and QPS algorithm, but also bring up a new graph-based software watermark framework with three graph encoding algorithms. The procedures of segmentation and recombination of graphs make watermarks harder to be detected. For being adopted to the process of graph in framework, we proposed *Link encoding*, *Color encoding* and *Link with color encoding* algorithm to increase robust and bit rate respectively. We also proposed a kind of method, path analysis, can be used in embedding phase of graph encoding, recovery from error or attack. Besides, we do some analysis and comparison in bit rate with each graph encoding to exam the resilience.

1.3 Synopsis

In the next chapter, we will introduce the related work of software watermark framework and algorithms. The proposed framework with three graph encoding will be expounded in Chapter 3. Analysis and comparison will be applied in Chapter 4 and 5 respectively. Finally, we give the conclusion in Chapter 6.

Chapter 2

Related Work

In this chapter, related software watermark algorithms will be introduced. At first, a software watermark framework which can adapt different graph encodings is published by Ramarathnam Venkatesan [5]. Static and dynamic software watermark algorithms both have each related graph-based encodings. QP algorithm is a significant concept to embed the watermark through graph-based encoding. QPS algorithm use color modification to improve QP algorithm. And CT algorithm provides a runtime execution algorithm to embed or recognize watermark.

2.1 Graph Theoretic Approach for Software Watermarking

Graph Theoretic Approach which is proposed by Venkatesan, Vazirani, and Sinha [5] provides a tool for software tamper resistance and against the graph based attack. In this approach, weak connection means that a link or function call between program P and watermark W is only a single edge between two subgraphs which are parsed from P and W . To prevent being identified as weak connection, graphs are efficiently separated into subgraphs which will be merged by adding edges, and graphs will be well connected. Subgraphs of W must be locally indistinguishable from P . Based on well connection and locally indistinguishableness. The steps of graph algorithm are shown as Figure 2.1. For given program P , watermark code W , secret keys ω_1 , ω_2 and ω_3 , and integer m :

Graph step: Flow graph G which has basic block as nodes and control flow or function calls as edges is computed from P . Similarly for W . G and W are both digraphs.

Clustering step: Using ω_1 as random seed to partition G into n clusters, so that edges straddling across clusters are minimized. Let G_c be the graph where each node corresponds to a cluster in G and there is an edge between two nodes if the corresponding clusters in G have an edge going

across them. W_c is yielded in the same way as G_c to produce undirected graphs of small order.

Merging step: Edges are added between G_c and W_c using a random process. The edges are added by a random process: when the node is v , the current values are d_{gg} and d_{gw} , the number of nodes adjacent to v in G_c and W_c respectively. Let $P_{gg} = d_{gg} / (d_{gg} + d_{gw})$ and $P_{gw} = d_{gw} / (d_{gg} + d_{gw})$. The next random node in G_c will be visited with probability P_{gw} or a node in W_c with probability P_{gg} and secret key ω_2 will make the choices. An edge is added between node v and its next random node. Repeat the step until the resultant graph H is yielded.

Recovery step: Finally, W_c is computed and encrypted with secret key ω_3 .

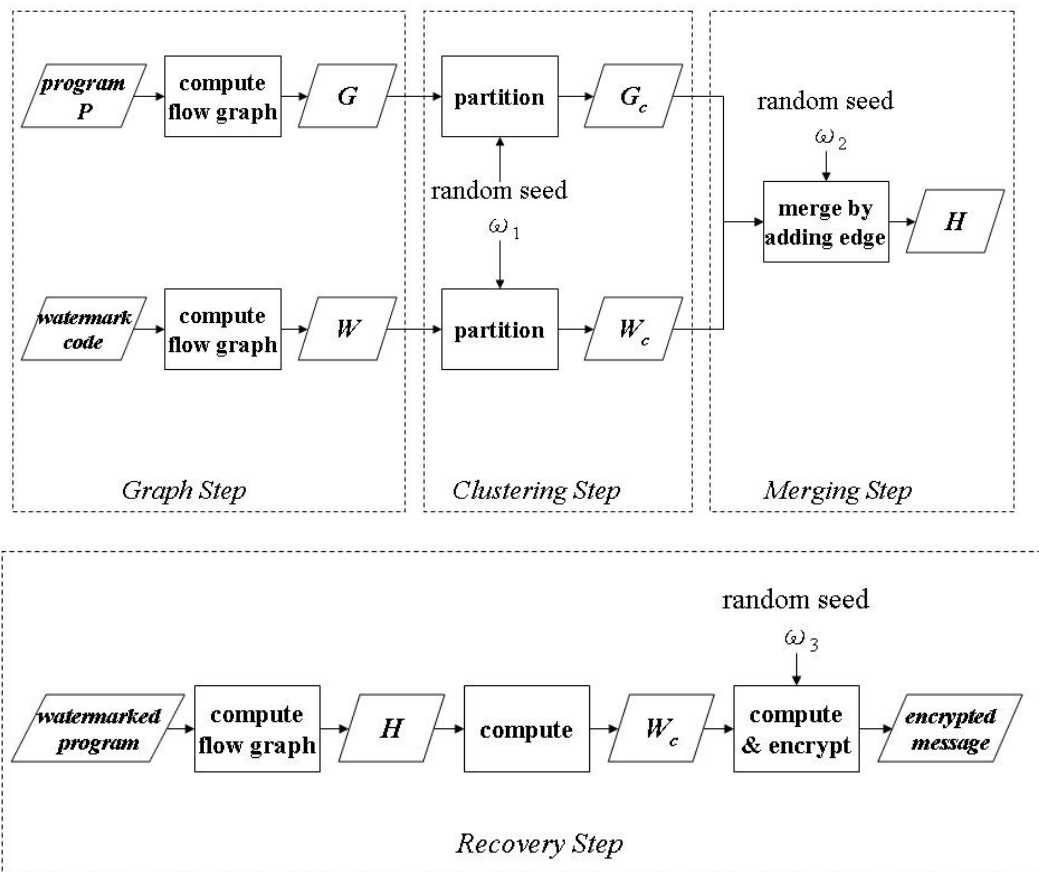


Figure 2.1 Graph Theoretic Approach

2.2 Static Watermark Algorithms

In static watermark algorithm, watermarks are stored in the application executable itself. Static

watermark may exist as code or data stored in the section of the program. QP and QPS are classified into this group.

2.2.1 QP Algorithm

Qu and Potkonjak have proposed QP algorithm for embedding a watermark. QP algorithm contains three kinds of watermark algorithms and *adding edges* is the algorithm we choose to study and improve. In this paper, for a given graph $G (V, E)$ and a message M to be embedded in G . Let vertices set $V = (v_0, v_1 \dots v_{n-1})$, edges set E and the message is encrypted into a binary string $M = m_0 m_1 \dots$ (By stream ciphers, block ciphers or cryptographic hash functions).

Embedding phase: First, a vertex v_i is selected from given graph $G (V, E)$ and find the nearest two vertices v_{i1} and v_{i2} for all $i < i_1 < i_2 \pmod n$ which are not connected to v_i , where means $(v_i, v_{i1}), (v_i, v_{i2}) \notin E$. And the rule for embedding according to m_i is as follows:

If $m_i = 0$, (v_i, v_{i1}) is put into E' , means the edge between v_i and v_{i1} is added.

If $m_i = 1$, (v_i, v_{i2}) is put into E' , means the edge between v_i and v_{i2} is added.

After the message $M = m_0 m_1 \dots$ are entirely embedded, a new graph $G' (V, E')$ which have new edges set is reported. For example, in Figure 2.2, a message $M = 5_{10} = 101_2$ has been embedded into a 6 vertices graph by 3 edges and each edge presents one bit of message M . The essence of this algorithm is to add an extra edge between two vertices, and these two vertices have to be colored by different colors.

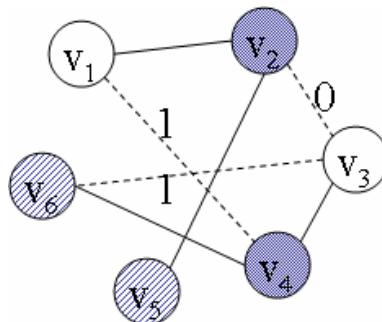


Figure 2.2 An Example of QP Algorithm

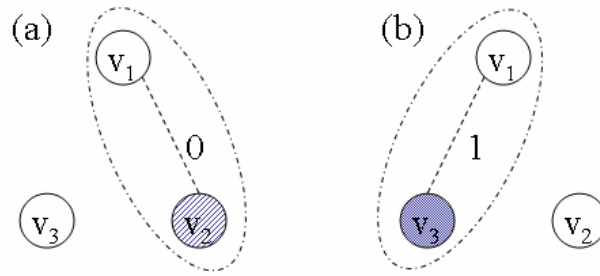


Figure 2.3 QP Recognition Algorithm

Recognition phase: In given graph $G' (V, E')$, each (v_i, v_j) is the vertices pair that one bit of the embedded message can be obtained. For each (v_i, v_j) , $j > i \pmod n$, the bit extraction is done by examining the number of vertices between v_i and v_j are not connected to v_i . There will be three cases to consider:

Case I : If there is no vertex which is not connected to v_i , a 0 bit will be extracted. The example is shown as Figure 2.3 (a).

Case II : If there is only one vertex which is not connected to v_i , a 1 bit will be extracted. The example is shown as Figure 2.3 (b).

Case III: If there is more than one vertex which is not connected to v_i , then reverse the order of v_i and v_j and repeat the extraction process.

2.2.2 QPS Algorithm

Myles and Collberg pointed out the error in QP algorithm, recognition failure. They provide two example of recognition failure as follow:

Example 1: Consider a graph $G (V, E)$ as Figure 2.4 (a) and the message $M = m_1m_2$ is 00. The embedding phase is illustrated as Figure 2.3 (b). At first, $v_i = v_0$, $v_{i1} = v_2$ and $v_{i2} = v_3$ are selected to embed $m_1 = 0$ by adding edge between v_0 and v_2 . And $v_i = v_3$, $v_{i1} = v_0$ and $v_{i2} = v_1$ are selected to embed $m_2 = 0$ by adding edge between v_3 and v_0 . New graph $G' (V, E')$ is obtained as Figure 2.3 (c).

In recognition phase, $m_1 = 0$ is found by examining the number of vertices not connected to v_0 between v_0 and v_2 . And $m_2 = 1$ is found by examining the number of vertices not connected to v_3

between v_0 and v_3 . The inaccurate message 01 is extracted when 00 was the embedded message.

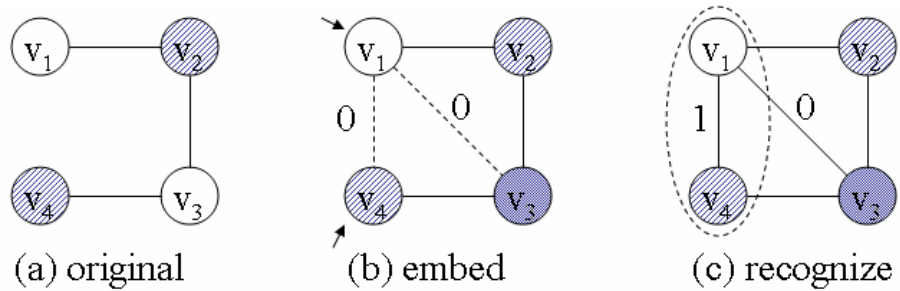


Figure 2.4 Failure Recognition of QP Algorithm

Example 2: When we embed the message 101 in the graph $G(V, E)$ as Figure 2.5 (a), the new graph $G'(V, E')$ is obtained as Figure 2.5 (b). By following the recognition algorithm, the message 1001 is recovered.

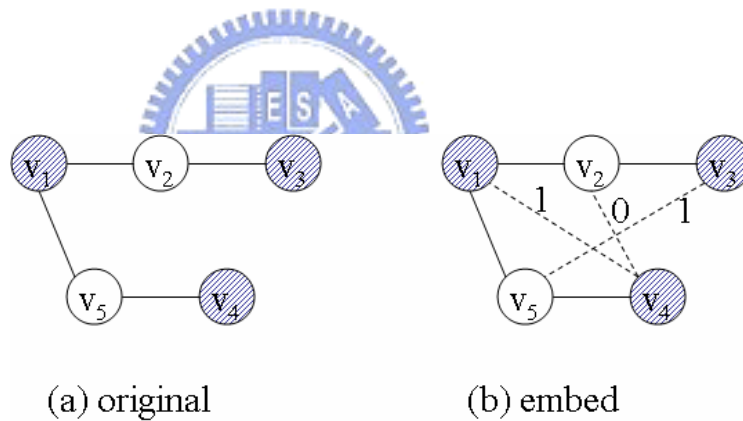


Figure 2.5 Failure Recognition of QP Algorithm

They considered the unpredictability of coloring for the vertices as the inaccurate message recognition of QP algorithm. To eliminate the unpredictability, QPS algorithm places additional constraints on which vertices can be selected for a triple. In this paper, for a given a graph $G = (V, E)$, a set of three vertices $\{v_1, v_2, v_3\}$ is considered a triple if

1. $v_1, v_2, v_3 \in V$,
2. $(v_1, v_2), (v_1, v_3), (v_2, v_3) \notin E$

And for a given a n -colorable graph $G = (V, E)$, a set of three vertices $\{v_1, v_2, v_3\}$ is considered a

colored triple if

1. $v_1, v_2, v_3 \in V$,
2. $(v_1, v_2), (v_1, v_3), (v_2, v_3) \notin E$, and
3. $\{v_1, v_2, v_3\}$ are all colored the same color.

Embedding phase: Select a vertex v_i which is not must already in a triple. Find the nearest two vertices v_{i1} and v_{i2} which are the same color as v_i and not already in a triple. An additional register allocator would be used to record related color of selected triple as (v'_i, v'_{i1}, v'_{i2}) . And the rule for embedding according to m_i is as follows:

If $m_i = 0$, add edge (v_i, v_{i1}) and change the color (v'_i, v'_{i1}) .

If $m_i = 1$, add edge (v_i, v_{i2}) .

The key idea of QPS embedding algorithm is to select the triples so that they are isolated units that will not affect other vertices in the graph. In addition, they use a specially designed register allocator which only changes the coloring of one of the two vertices involved in the added edge and no other vertices in the graph.

Recognition phase: QPS recognition algorithm works by identifying triples which had been selected in the embedding phase. A triple (v_i, v_{i1}, v_{i2}) has been identified, and its related *colored triple*, (v'_i, v'_{i1}, v'_{i2}) , is examined. If v'_i and v'_{i1} are different color, a 0 was embedded, otherwise a 1 was embedded.

2.2.3 Problems

The colors of vertices provide starting-point to discuss. In QP embedding algorithm, an extra edge is added between two vertices and these two vertices which may not be necessary in the original graph G will be colored by different colors. In QP recognition algorithm, we will find colors between two vertices are not accurate feature to observe one bit of message had been embedded.

The coloring feature in QP algorithm is ambiguous. This results in bad performance in embedding and recognizing watermark information.

In Example 1, when $v_i = v_3$, $v_{i1} = v_0$ and $v_{i2} = v_1$ are selected to embed $m_2 = 0$, QP embedding algorithm which is defined as $i < i_1 < i_2 \pmod n$ has been mistaken about. Unpredictable error could occur when we embed the message and neglect the definition $i < i_1 < i_2 \pmod n$. For illustration, as Figure 2.6, the message $M = 1$ will be embedded into original graph, as Figure 2.6 (a). In embedding phase, $v_i = v_2$, $v_{i1} = v_3$ and $v_{i2} = v_0$ are selected to embed $M = 1$ as Figure 2.6 (b). In recognition phase, the message will be identified by examining the number of vertices not connected to v_0 between v_0 and v_2 and there are two cases can take into consider:

- Case 1: $v_i = v_3$ is considered that is not connected to $v_i = v_0$ and a 1 message is identified.
- Case 2: $v_i = v_1$ is considered that is not connected to $v_i = v_0$ and a 0 message is identified.

Different message is identified when different vertex is taken into consider and it is uncertain whether message is true. This makes it clear that Example 1 is not a proper example to illustrate problem of QP algorithm.

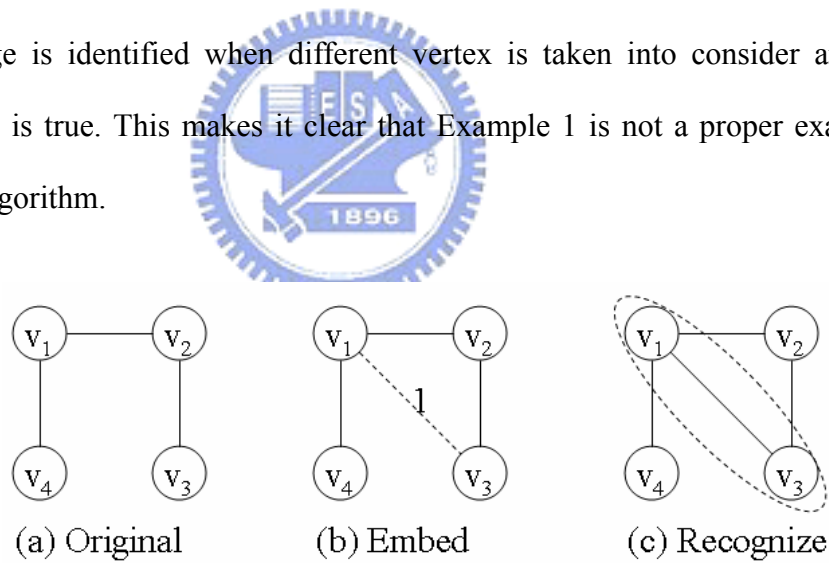


Figure 2.6 Example of Recognition Problem in QP Algorithm

Some problems occur when the definition is followed already. To illustrate, Example 3 and 4 are applied to consider and there is no need to go into details about the color between vertices:

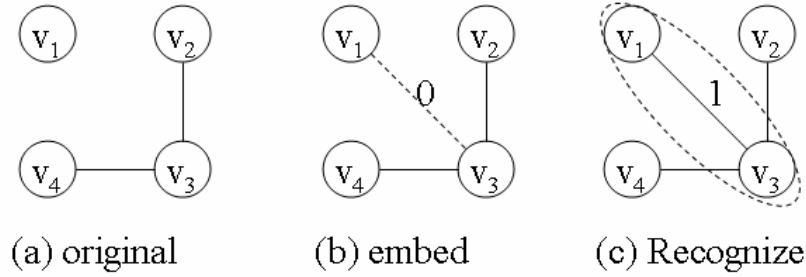


Figure 2.7 Example of Recognition Problem in QP Algorithm

Example 3: Consider a graph $G (V, E)$ as Figure 2.7 (a) and the message $M = 0$. The embedding phase is illustrated as Figure 2.7 (b), and $v_i = v_0$, $v_{i1} = v_2$ and $v_{i2} = v_3$ are selected to embed $M=0$ by adding edge between v_0 and v_2 . New graph $G' (V, E')$ is obtained as Figure 2.7 (c). In recognition phase, $M = 1$ is found by examining the number of vertices not connected to v_0 between v_0 and v_2 . The inaccurate message 1 is extracted when 0 was the embedded message.

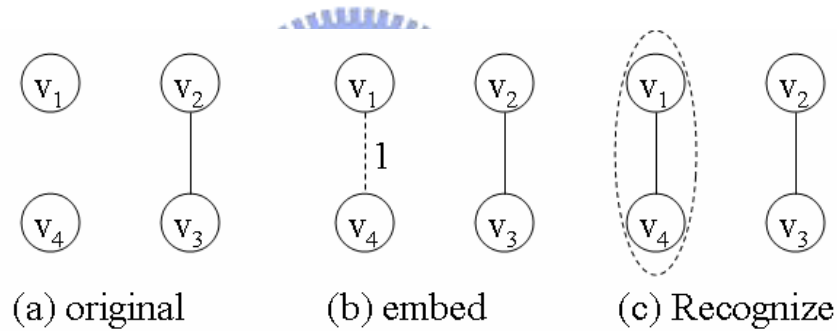


Figure 2.8 Example of Recognition Problem in QP Algorithm

Example 4: Consider a graph $G (V, E)$ as Figure 2.8 (a) and the message $M = 1$. The embedding phase is illustrated as Figure 2.8 (b), and $v_i = v_0$, $v_{i1} = v_2$ and $v_{i2} = v_3$ are selected to embed $M=0$ by adding edge between v_0 and v_3 . New graph $G' (V, E')$ is obtained as Figure 2.8 (c). In recognition phase, the number of vertices not connected to $v_i = v_0$ between $v_i = v_0$ and $v_j = v_3$ is 2, and the same we reverse the order as $v_i = v_3$ and $v_j = v_0$. It is an undefined case that an unknown message is extracted if the number is 2.

The cause of the problem can be traced back to the assumption of embedding phase of QP algorithm: find the nearest two vertices v_{i1} and v_{i2} which are not connected to v_i . We define the graph have

“vertices pair” which means the nearest vertices v_{i1} and v_{i2} are not connected to v_i . When v_i is selected, there are many vertices pairs $\{v_{i1}, v_{i2}\}$ can be selected to embed the message. In Figure 2.6, when $v_i = v_0$, there are two vertices pairs, $\{v_1, v_2\}$ and $\{v_2, v_3\}$ for being selected to embed the message. Correct message will be extracted if $\{v_1, v_2\}$ is selected. In Example 3, an inaccurate 1 message is extracted when $\{v_2, v_3\}$ is selected. In the same way, in Figure 2.8, correct message will be extracted if $\{v_1, v_2\}$ is selected and inaccurate message will be extracted if $\{v_2, v_3\}$ is selected.

Take a more carefully look into the selection of vertices pairs in the embedding phase and we find the result: Only if the vertices pair is the nearest to v_i is restricted to selected, the message must be correct is extracted. This restrict can be treated as the definition in QP embedding phase.

2.3 Dynamic Watermark Algorithm

The basic concept of dynamic watermark algorithm is to embed watermark information to the running state of a program. In the other words, the watermark information can be embedded and extracted at runtime and thus make the disclosure more difficult. However, the implementation of dynamic watermark algorithm is difficult for most programming languages. Generally, Java-based language can implement algorithm easily. It can divide into four phases:

Annotation: Adding annotation (or mark) points into the application to be watermarked before the watermark can be embedded. Functions will be inserted into these annotation points. These functions perform no action and simply indicate to the locations where watermark can be inserted. Locations are preferred mark locations that allocate objects and manipulate pointers and directly depend on user input. Hot spots and non-deterministically execution should avoid mark locations.

Tracing: A tracing run of the program will be performed after the application has been annotated. Some annotation points will be selected after the tracing run. These annotation points will be the location where watermark-building code will later be inserted.

Embedding: Hence, embedding of watermark will start when the application has been traced. The input will be converted into an integer. From the integer, a graph G is generated to embed. The

embedding of watermark is divided into four steps:

1. Watermark W is embedded by generating graph G .
2. Generating an *intermediate code* to build this graph and translate into a Java method.
3. Replace the functions in annotation phase with these new functions.
4. New Java method file will be executed to build the graph G on the heap.

A single graph encoding method is not expected to fill requirements (high bit rate, high resilience to attack, etc.) in CT algorithm. Develop a library of graph encoding for building watermark graphs which have different characters instead. There are five kinds of graph encoding methods are implemented: Permutation encoding · Radix encoding · Parent-pointer trees · Reducible permutation graph and Planted plane cubic trees.

Extraction: In watermarked application, extraction is run as a sub-process under debugging. The secret input sequences are exactly entered as input and same mark allocations will be hit during tracing run. If the last part of input has been entered, the heap is examined for graphs that could potentially be a complete watermark graphs. Number of watermark will be extracted from graphs and reported.

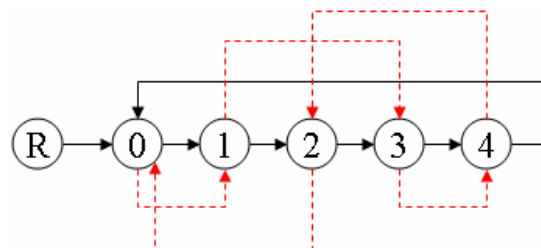


Figure 2.9 Example of Permutation Encoding Algorithm

Permutation Encoding: A watermark integer in the range $[0...n-1]$ can be shown by permuting the

numbers $\langle 0, \dots, n-1 \rangle$, the mapping of permutation will represent a number. For an example, the original $A = \langle 0, 1, 2, 3, 4, 5, 6, 7 \rangle$ and a watermark is 1000, the permuted A will be $A = \langle 0, 7, 2, 1, 4, 3, 6, 5 \rangle$ to represent the watermark. Permutation can be constructed by a singly-linked, circular list data structure, such as Figure 2.9.

Radix Encoding: A Radix graph is a circular linked list with n lengths, order of node represent the number of exponent with base- n digit and data pointer is the coefficient. A null-pointer encodes a 0, a self-pointer is a 1, and a pointer to next node encodes a 2, and so on. Take Figure 2.10 as an example, $3 \times 6^4 + 2 \times 6^3 + 3 \times 6^2 + 4 \times 6^1 + 1 \times 6^0$ can be represented by the graph.

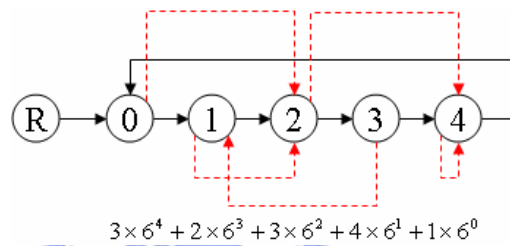


Figure 2.10 Example of Radix Encoding Algorithm

Parent-pointer trees encoding (PP trees): This encoding algorithm can be described as enumerations of graphs. The idea is that the watermark number n can be represented by the index of the watermark graph in a table of enumeration and the number of nodes depends on your watermark. Figure 2.11 is an example which represents the number 1, 2, 20 and 21.

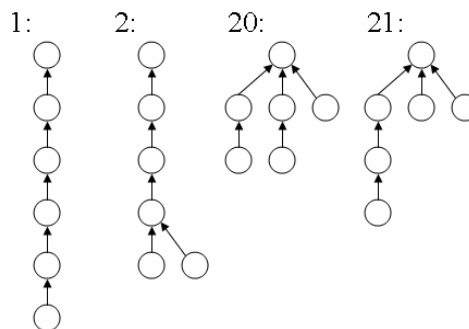


Figure 2.11 Example of PP Tree Encoding Algorithm

2.4 Summary

In this chapter, we introduce advantage and disadvantage of some famous framework and algorithms. Based on these characters, we will design and modify our proposed software watermark framework and algorithms.



Chapter 3

The Proposed Software Watermarking Framework

As described in the previous chapter, we explain the problems of failure recognition, low bit rate and color in QP and QPS algorithm. We also proposed static software watermark algorithms by leveraging the concepts in CT dynamic algorithm. The proposed static watermark algorithms will make the improvement on these characters:

- Easier in constructing and extracting
- Higher bit rates
- Better robust from common attacks
- More programming languages

We proposed a flexible framework to adopt not only proposed graph encoding algorithms but also other graph algorithms. To construct and extract watermark easily, we modify the heavy procedure of random walk in *Graph Theoretic Approach*. It is also implemented easily with more kinds of programming languages.

3.1 Framework

Framework can be divided into two phases: embedding and recognition phases. Embedding and recognition phase are shown with Figure 3.1 and Figure 3.3 separately.

In embedding phase, process proceeds through three steps:

Transformation step: Program P is parsed into graph G which is constructed by vertices and edges. Each vertex of graph represents a basic block consisting of instructions. Each edge will be a directed edge represents a function call between two basic blocks. Order of vertices is arranged by depth-first search (DFS) algorithm. Message M can be a statement or special integer in binary

format. To make binary conversion, the hash function or other transform function can also be adopted to achieve higher security.

To increase the complexity and improve the privacy, graph G is segmented into n subgraphs, $\{g_1, g_2, \dots, g_n\}$, where g_i is one of the subgraph of G . The representation of the g_i shown as $G = \{g_i | 1 \leq i \leq n\}$, where n is the number of subgraphs. Message M is also segmented into n fragmental messages, $\{m_1, m_2, \dots, m_n\}$, using the random seed ω , where m_i is one of the fragmental messages of M . The representation of the m_i shown as $M = \{m_i | 1 \leq i \leq n\}$, where n is the number of subgraphs. Each subgraph is constructed according to the *vertices selecting rule* that make each subgraph which is decided by the corresponding fragmental message being embedded successfully.

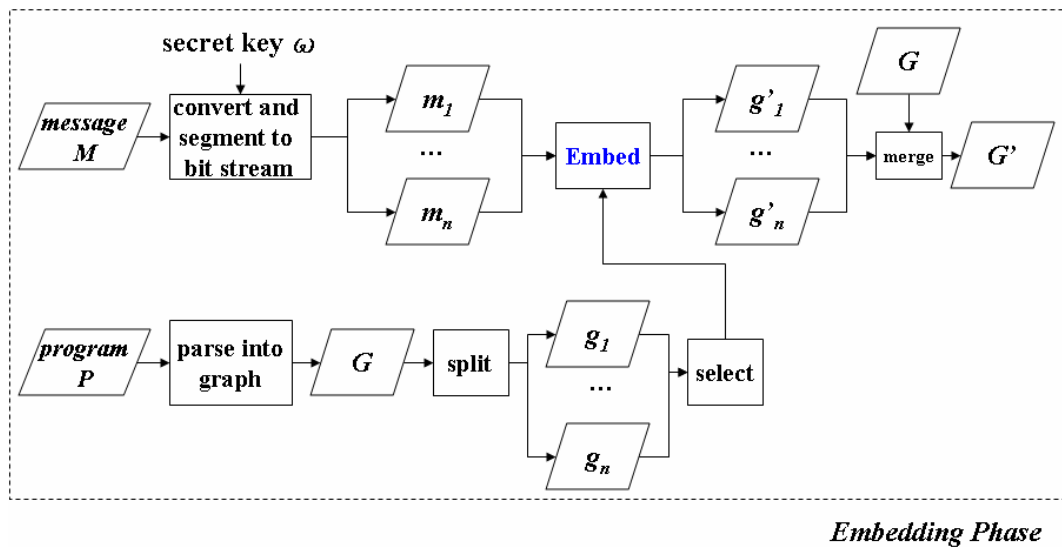


Figure 3.1 Embedding Phase of Proposed Framework

The *vertices selecting rule* is defined as follow:

1. Select the vertex in the higher level of sub-graph as the first-selected vertex.
2. Select the vertices which aren't the children of first vertex of sub-graph.

Each subgraph should have these characters:

1. The size of subgraph is decided by corresponding fragmental message
2. A sub-graph should contain at least three vertices.
3. If sub-graph is constructed from k vertices, fragmental message should be $k-2$ bits.

Embed step: When fragmental messages and subgraphs are generated, the next move is selecting the algorithm of graph encoding. There are three kinds of graph encoding algorithms: link encoding (LE) · color encoding (CE) and link color encoding (LCE). If LE or LCE is selected, path analysis will proceed. During the process of path analysis, as Figure 3.2, subgraph and the path which has more levels for embedding is analyzed by tree diagram. First vertex is selected according to the result from path analysis. Follow on the first vertex, fragmental message is embedded into its corresponding subgraph by adding directed edge between two vertices.

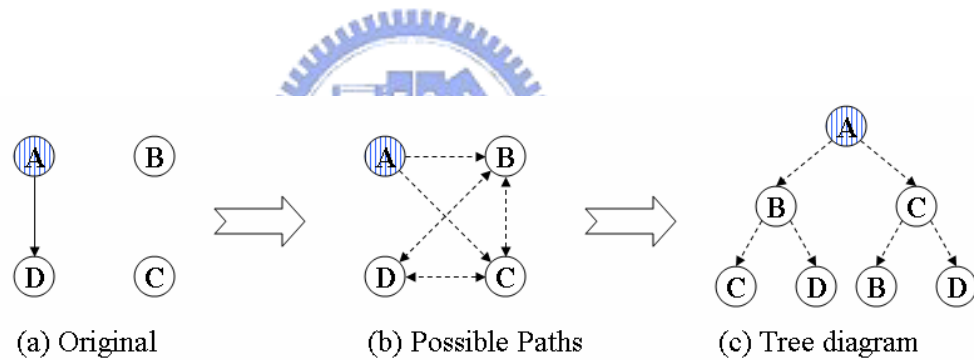


Figure 3.2 Example of Path Analysis

Merge step: The last step is merging original graph G and each subgraph into new graph. Directed edge is added between two vertices in original graph if the same vertices in subgraph have been embedded a bit of fragmental message. Color of vertices in original graph will be tampered according to the same vertices in subgraph. Finally, a new graph G' contains the embedded message is generated.

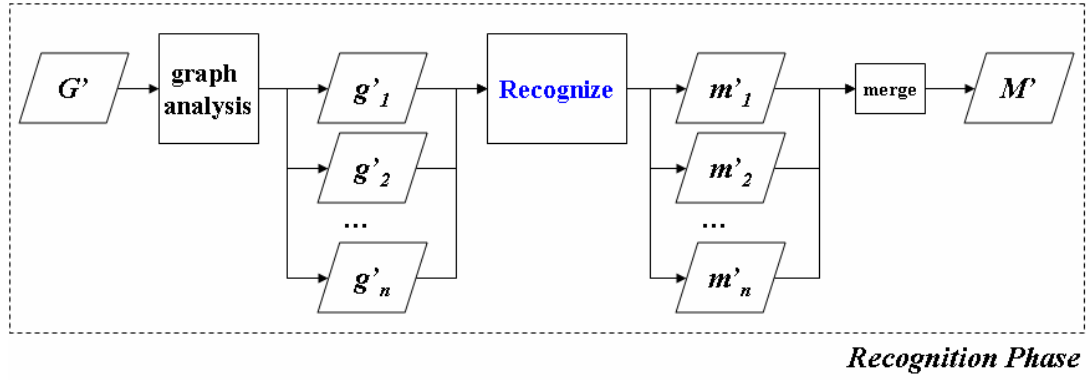


Figure 3.3 Recognition Phase of Proposed Framework

In recognition phase, there are two steps:

Recognition step: When watermarked graph G' is received, information of subgraphs, vertices and its corresponding color, and edges are also known. Algorithm of graph encoding which was used to embed the message is also analyzed and found. In the light of information, graph G' is segmented into n subgraphs $\{g'_1, g'_2, \dots, g'_n\}$, where g'_i is one of the subgraph of watermarked graph G' . The representation of the g_i is shown as $G' = \{g'_i \mid 1 \leq i \leq n\}$, where n is the number of subgraphs. According to that graph encoding recognition algorithm, the fragmental messages $\{m'_1, m'_2, \dots, m'_n\}$ are extracted from each subgraph, where m'_i is one of message M' . The representation of the m_i is shown as $M' = \{m'_i \mid 1 \leq i \leq n\}$, where n is the number of subgraphs.

Message processing step: Combine each fragmental message into new message m' . It is not necessary that new message M' must be equal to original message M' if we can verify the correct hidden information from M' .

Figure 3.4 is an example of proposed framework in embedding phase:

1. Input is $m = 1011$ and program P .
2. P is parsed into graph G , and index is arranged by DFS algorithm.
3. M_B is segmented into two fragmental messages $m_1 = 10$ and $m_2 = 11$. G is also segmented into two subgraphs g_1 and g_2 with *vertices selecting rule*.

4. When *link encoding* algorithm is selected, each subgraph is analyzed by path analysis and the pivot vertex is decided. Then, fragmental messages b_{i1} and b_{i2} are embedded into subgraphs g_1 and g_2 .
5. Merge graph G and subgraphs g_1 and g_2 into new graph G' .

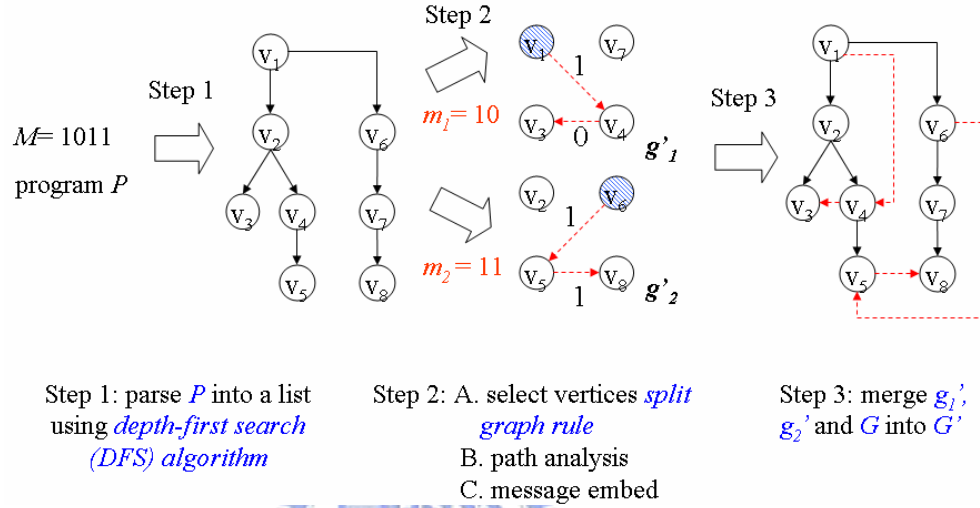


Figure 3.4 Example of Proposed Embedding Phase Framework

3.2 Proposed Graph Encoding Algorithms

In this section, we will introduce proposed graph encoding algorithms.

At first, the definition is as follows:

Given fragmental message m_i , represent as $\{b_{i1}, b_{i2}, \dots, b_{in}\}$, where $m_i = \{b_{ij} | 1 \leq j \leq \beta\}$, β is number of bit in fragmental message and a subgraph $G_i(V_i, E_i)$ which contains vertices set V_i and edges set E_i . Vertices pair, $\{v_{a1}, v_{a2}\}$ contains nearest two vertices v_{a1} and v_{a2} , and is also the nearest vertices pair not connected to v_a . Edges $(v_a, v_{a1}), (v_a, v_{a2})$ are not in edges set E_i .

3.2.1 Link Encoding (LE)

This algorithm can have improvement in robust by the way of link list. *Pivot vertex* is the vertex which is selected according to *path analysis* [24] [22] and subgraph will have the higher capacity

for embedding message.

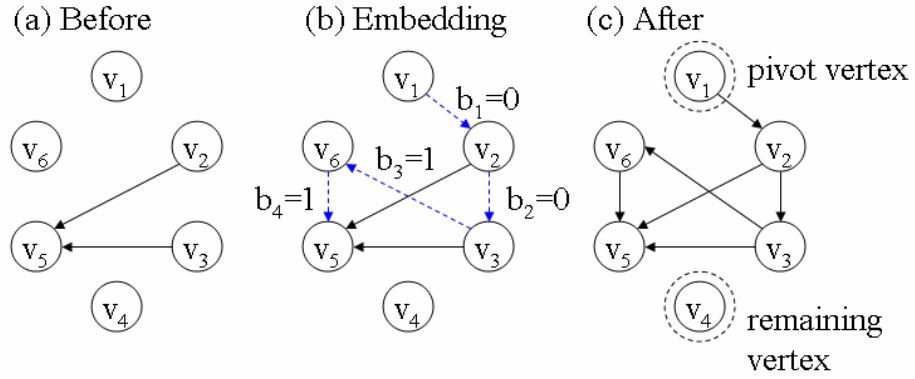


Figure 3.5 Example of Link Encoding in Embedding Phase

```

Input: watermark  $m=b_1b_2\dots b_n$  and a graph  $G(V,E)$ 
Output:  $G'(V, E')$ , pivot vertex  $v_p$ , remaining vertex  $v_r$ 
Pseudo Code:
     $v_p = \text{path\_analyze}(g_i, n);$ 
    if  $v_p$  is not found
        return NULL;
     $v_a = v_p;$ 
     $V' = V;$ 
    foreach bit  $b_j$ 
    {
        search  $V'$  and find the vertices pair  $(v_{a1}, v_{a2})$  that are nearest but not connected to  $v_a$ 
         $V' = V' - \{v_a\};$ 
        if  $b_j=0$ 
             $E' = E \cup (v_a, v_{a1});$ 
             $v_a = v_{a1};$ 
        else
             $E' = E \cup (v_a, v_{a2});$ 
             $v_a = v_{a2};$ 
    }
     $v_r = \text{the last element in } V';$ 
    return  $G'(V, E');$ 

```

Figure 3.6 Pseudo Code of LE Embedding Algorithm

Embedding phase: v_a is selected *pivot vertex*. For the first message b_{i1} , vertices pair $\{v_{a1}, v_{a2}\}$ is the nearest vertices pair not connected to v_a is found. If $b_{i1} = 0$, directed edge (v_a, v_{a1}) is added. Else, $b_{i1} = 1$, directed edge (v_a, v_{a2}) is added. Then, the vertex v_a is treated as invisible vertex and its connected vertex (v_{a1} or v_{a2}) is treated as new *pivot vertex* which will be embedded with the next message m_i . Repeat the step until $m_i = b_{i1} b_{i2} \dots b_{in}$ are all embedded into subgraph $G_i(V_i, E_i)$ and

new graph $G'_i(V'_i, E'_i)$ will be generated. We will find that the last one vertex which is not used during the embedding step and the information of *pivot vertex* and *remaining vertex* is useful for robust. Figure 3.5 is an example of link encoding in embedding phase. For given subgraph, as Figure 3.5 (a), $m_i = 0011$ and *pivot vertex* v_1 is selected. First directed edge (v_1, v_2) is added according to $b_{i1} = 0$ and $\{v_2, v_3\}$ is the nearest vertices pair not connected to v_1 . v_2 is treated as new *pivot vertex* and v_1 is treated as invisible vertex when $b_{i2} = 0$ is ready for embedding. Repeat the step as Figure 3.5 (b) until $m_i = b_{i1} b_{i2} b_{i3} b_{i4}$ are embedded. The pseudo code for the embedding phase in LE algorithm is described in Figure 3.6.

```

Input: watermarked graph  $G'(V', E')$ , pivot vertex  $v_p$ , remaining vertex  $v_r$ 
Output: watermark  $m=b_1b_2\dots b_n$ 
Algorithm:
   $V' = V$ ;
   $v_a = v_p$ ;
   $V' = V' - \{v_a\}$ ;
  foreach  $j$  between 1 and  $n$ 
  {
    foreach vertex  $v_k$  in  $V'$ 
      if  $v_k$  is adjacent to  $v_a$ 
        count = 0;
        foreach  $q$  between  $k$  and  $a$ 
          if  $v_q$  is not connected to  $v_a$ 
            count ++;
        if count == 0
           $b_j = 0$ ;
           $v_a = v_k$ ;
        elseif count == 1
           $b_j = 1$ ;
           $v_a = v_k$ ;
        else
          continue; //check next adjacent vertex of  $v_a$ 
     $V' = V' - \{v_a\}$ ;
  }
   $v_r' =$  the last element in  $V'$ ;
  if  $v_r' = v_r$ 
    return  $m$ ;
  else
    return NULL;

```

Figure 3.7 Pseudo Code of LE Recognition Algorithm

Recognition phase: How can we extract the message from graph? Given the graph $G'_i(V'_i, E'_i)$ and the information of *pivot vertex* v_a , find the number of vertices not connected to v_a between v_a and its connected vertex. If the number is zero, b_{i1} is 0, and if the number is 1, b_{i1} is 1. For the vertex

connected to v_a and treating v_a as an invisible vertex, the next message will be extracted. Repeat the step until that there is only one vertex in subgraph and compare this vertex with *remaining vertex*. We will make sure the message is correct if the answer is “the same”. The pseudo code for the recognition phase in LE algorithm is described in Figure 3.7.

3.2.2 Color Encoding (CE)

The function of color is applied for increasing bit rate in color encoding algorithm. All the vertices in subgraph are the same color in original. The color rule is defined as follows:

v_a and its connected vertex v_b are same color, a 00 message is embedded.

v_a and its connected vertex v_b are different color, a 01 message is embedded.

```

Input: watermark  $m=b_1b_2...b_n$ , a graph  $G(V,E)$ 
Output:  $G'(V, E')$ , vertex color set  $C = \{c_1, c_2, ...c_n\}$  and an embedding vertex set  $V_x$ .
Algorithm:
  C = initialize_vertex_colors(V);
  V' = V;
  Vx =  $\phi$ ;
  foreach  $b_jb_{j+1}$ 
  {
     $v_a = \text{smallest}(V')$ ;
     $v_{a1}, v_{a2} = \text{closest\_vertices\_pair}(v_a)$ ;
    switch ( $b_jb_{j+1}$ ) {
      case 00:
         $E' = E' \cup (v_a, v_{a2})$ ;
        break;
      case 01:
         $E' = E' \cup (v_a, v_{a2})$ ;
         $c_{a2} = \text{new color different than } c_{a2}$  //change  $v_{a2}$ 's color;
        break;
      case 11:
         $E' = E' \cup (v_a, v_{a1})$ ;
        break;
      case 10 :
         $E' = E' \cup (v_a, v_{a1})$ ;
         $c_{a1} = \text{new color different than } c_{a1}$  //change  $v_{a1}$ 's color;
        break;
    }
     $V' = V' - \{v_a\}$ ;
     $V_x = V_x + \{v_a\}$ ;
  }
  return  $G'(V, E'), C'$ ;

```

Figure 3.8 Pseudo Code of CE Embedding Algorithm

Embedding phase: For $m_i = b_{i1} b_{i2} ... b_{in}$, each fragment message is a 2-bit message in this method.

Find vertices pair $\{v_{a1}, v_{a2}\}$ that are nearest pair and not connected to v_a .

$b_{ij} b_{i(j+1)} = 00$ (v_a, v_{a2}) is added, v_a and v_{a2} are same color.

$b_{ij} b_{i(j+1)} = 01$ (v_a, v_{a2}) is added, v_a and v_{a2} are different color.

$b_{ij} b_{i(j+1)} = 11$ (v_a, v_{a1}) is added, v_a and v_{a1} are same color.

$b_{ij} b_{i(j+1)} = 10$ (v_a, v_{a1}) is added, v_a and v_{a1} are different color.

The new graph $G'_i (V'_i, E'_i)$ which contains new edges set E'_i and new vertices set V'_i with its related information of color is generated. The pseudo code for the embedding phase in CE algorithm is described in Figure 3.8.

Figure 3.9 is a simple example of color encoding in embedding phase. At first, $m_i = 1101$ is segmented into 11 and 01. v_1 is selected as v_a , and $\{v_2, v_3\}$ is the nearest vertices pair that are not connected to v_1 . For the first two bits 11, the directed edge (v_1, v_2) is added and v_1 and v_2 are still same color. And the next message, v_2 is selected as v_a , and $\{v_3, v_4\}$ is the nearest vertices pair that are not connected to v_2 . For the next two bits 01, the directed edge (v_2, v_4) is added and v_2 and v_4 are colored by different color.

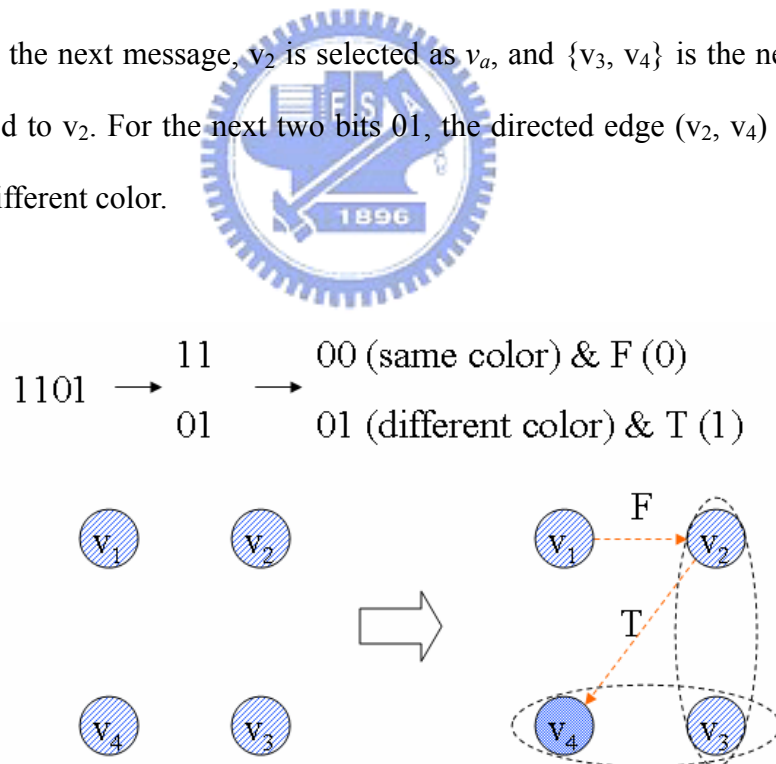


Figure 3.9 Example of Color Encoding in Embedding Phase

Recognition phase: Given the graph $G'_i (V'_i, E'_i)$ and vertices set with information of color. Find the number of vertices not connected to v_a between v_a and its connected vertex v_b . The embedded bit

is extracted according the rule as follows:

The number is 0, v_a and v_b are same color, then $m_i = 11$

The number is 0, v_a and v_b are different color, then $m_i = 10$

The number is 1, v_a and v_b are same color, then $m_i = 00$

The number is 1, v_a and v_b are different color, then $m_i = 01$

The pseudo code for the recognition phase in CE algorithm is described in Figure 3.10.

```

Input: a graph  $G(V, E)$ , an embedding vertex set  $Vx$  and its color set  $C$ 
Output:  $m=b_1b_2...b_n$ 
Algorithm:
do
{
   $V_a = \text{first element in } Vx;$ 
   $Vx = Vx - \{V_a\};$ 
  find the closest vertices  $V_{a1}, V_{a2}$  that are not connected to  $V_a$ ;
  count for the vertices whose indices are in between  $V_{a1}$  and  $V_{a2}$ , and are not connected to  $V_a$ ;
  switch (count)
  {
    case 0:
      if ( $V_a, V_{a1}$ ) exists in  $E$  and  $C_a == C_{a1}$ 
         $b_j b_{j+1} = 11;$ 
      elseif ( $V_a, V_{a1}$ ) exists in  $E$  and  $C_a != C_{a1}$ 
         $b_j b_{j+1} = 10;$ 
      elseif ( $V_a, V_{a2}$ ) exists in  $E$  and  $C_a == C_{a2}$ 
         $b_j b_{j+1} = 11;$ 
      elseif ( $V_a, V_{a2}$ ) exists in  $E$  and  $C_a != C_{a2}$ 
         $b_j b_{j+1} = 10;$ 
      break;
    case 1:
      if ( $V_a, V_{a1}$ ) exists in  $E$  and  $C_a == C_{a1}$ 
         $b_j b_{j+1} = 00;$ 
      elseif ( $V_a, V_{a1}$ ) exists in  $E$  and  $C_a != C_{a1}$ 
         $b_j b_{j+1} = 01;$ 
      elseif ( $V_a, V_{a2}$ ) exists in  $E$  and  $C_a == C_{a2}$ 
         $b_j b_{j+1} = 00;$ 
      elseif ( $V_a, V_{a2}$ ) exists in  $E$  and  $C_a != C_{a2}$ 
         $b_j b_{j+1} = 01;$ 
      break;
  }
} while ( $Vx$ )
return  $m$ ;

```

Figure 3.10: Pseudo Code of CE Recognition Algorithm

3.2.3 Link with Color Encoding (LCE)

The third method is link with color encoding, as implied in the name, and is combined with link encoding and color encoding. LCE method is successful to have the higher robust and bit rate.

Follow the definition of link encoding and color encoding, this method is introduced as follow:

Embedding phase: Given message $m_i = b_{i1} b_{i2} \dots b_{in}$ and a subgraph $G_i(V_i, E_i)$, v_a is selected *pivot vertex*. Find vertices pair $\{v_{a1}, v_{a2}\}$ is the nearest vertices pair not connected to v_a . First two bit of message $b_{i1} b_{i2}$ are embedded according to the rule as follows:

$b_{i1} b_{i2} = 00$ (v_a, v_{a2}) is added, v_a and v_{a2} are same color.

$b_{i1} b_{i2} = 01$ (v_a, v_{a2}) is added, v_a and v_{a2} are different color.

$b_{i1} b_{i2} = 11$ (v_a, v_{a1}) is added, v_a and v_{a1} are same color.

$b_{i1} b_{i2} = 10$ (v_a, v_{a1}) is added, v_a and v_{a1} are different color.

Then, the *pivot vertex* v_a is treated as invisible vertex and its connected vertex is treated as new *pivot vertex* which will be embedded with the next two bits $b_{i3} b_{i4}$. Repeat the step until $m_i = b_{i1} b_{i2} \dots b_{in}$ are all embedded into subgraph $G_i(V_i, E_i)$ and new graph $G'_i(V'_i, E'_i)$ will be generated. The pseudo code for the embedding phase in LCE algorithm is described in Figure 3.11. Figure 3.12 is an example with LCE method in embedding phase.



```

Input:  $G(V, E)$ ,  $m = b_1 b_2 \dots b_n$ , and color set  $C = \{c_1, c_2, \dots, c_n\}$ 
Output:  $G'(V, E')$ ,  $C'$ , the pivot vertex  $v_p$ , the remaining vertex  $v_r$ 
Algorithm:
  foreach  $b_j b_{j+1}$ 
  {
    select a pivot vertex  $v_p$  from  $V$ ;
    let  $v_a = v_p$ ;
    start from the vertex  $v_a$ , where  $a$  is the smallest vertex index in  $V$ ;
    find the closest vertices  $v_{a1}, v_{a2}$  that are not connected to  $v_a$ ;
    switch ( $b_j b_{j+1}$ )
    {
      case 00:
        add edge ( $v_a, v_{a2}$ ) to  $E$ ;
         $v_a = v_{a2}$ ;
        break;
      case 01:
        add edge ( $v_a, v_{a2}$ ) to  $E$ ;
         $c_{a2} =$  different color from  $c_a$ ;
         $v_a = v_{a2}$ ;
        break;
      Case 11:
        add edge ( $v_a, v_{a1}$ ) to  $E$ ;
         $v_a = v_{a1}$ ;
        break;
      Case 10:
        add edge ( $v_a, v_{a1}$ ) to  $E$ ;
         $c_{a1} =$  different color from  $c_a$ ;
         $v_a = v_{a1}$ ;
        break;
    }
  }

```

```

    }
  }
  return  $G'(V, E')$  and  $C'$ ;

```

Figure 3.11: Pseudo Code of LCE Embedding Algorithm

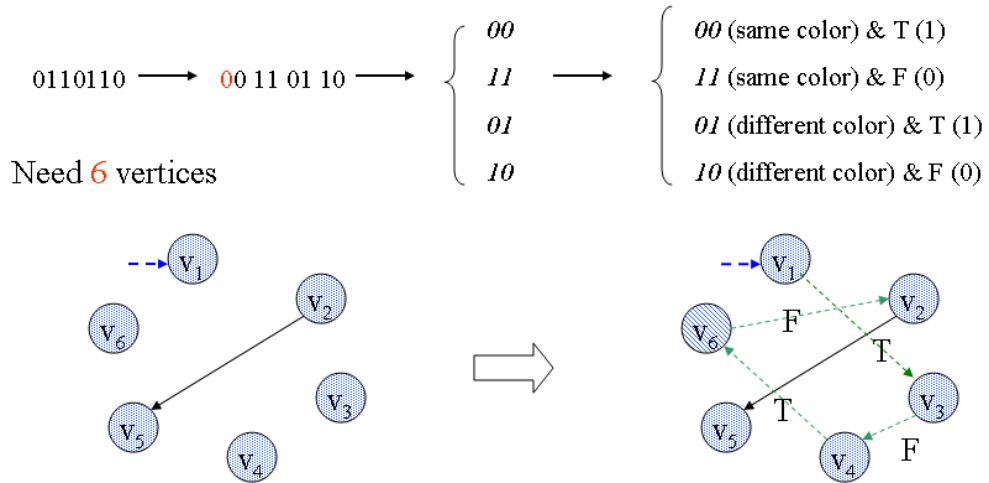


Figure 3.12: Example of Link with Color Encoding in Embedding Phase

Recognition phase: Given the graph $G'_i(V'_i, E'_i)$ and vertices set with information of color. Find the number of vertices not connected to v_a between v_a and its connected vertex v_b . The first embedded bits $b_{i1} b_{i2}$ are extracted according to the rule as follows:

- The number is 0, v_a and v_b are same color, then $b_{i1} b_{i2} = 11$
- The number is 0, v_a and v_b are different color, then $b_{i1} b_{i2} = 10$
- The number is 1, v_a and v_b are same color, then $b_{i1} b_{i2} = 00$
- The number is 1, v_a and v_b are different color, then $b_{i1} b_{i2} = 01$

With the vertex v_b connected to v_a and treating v_a as an invisible vertex, the next message m_l will be extracted. Repeat the step until that there is only one vertex in subgraph and compare this vertex with *remaining vertex*. We will make sure the message is correct if answer is “the same”. The pseudo code for the recognition phase in LCE algorithm is described in Figure 3.13.

```

Input: a watermarked graph  $G'(V, E')$ , the pivot vertex  $v_p$ , the remaining vertex  $v_r$  and the color set  $C$ 
Output:  $m = b_1 b_2 \dots b_n$ 
Algorithm:
  start from the pivot vertex  $v_p$ ;
  let visiting vertex  $v_a$  equals to  $v_p$ ;
  find the closest vertices  $v_{a1}, v_{a2}$  that are not connected to  $v_a$ ;
  foreach two bits  $b_j b_{j+1}$ 

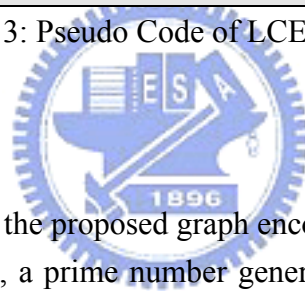
```

```

{
    count for the vertices whose indices are in between a1 and a2, and are not connected to va;
    switch (count)
    {
        case 0:
            if (va, va1) exists in E and ca == ca1
                bjbj+1=11;
            elseif (va, va1) exists in E and ca != ca1
                bjbj+1=10;
            elseif (va, va2) exists in E and ca == ca2
                bjbj+1=11;
            elseif (va, va2) exists in E and ca != ca2
                bjbj+1=10;
            break;
        case 1:
            if (va, va1) exists in E and ca == ca1
                bjbj+1=00;
            elseif (va, va1) exists in E and ca != ca1
                bjbj+1=01;
            elseif (va, va2) exists in E and ca == ca2
                bjbj+1=00;
            elseif (va, va2) exists in E and ca != ca2
                bjbj+1=01;
            break;
    }
}
return mi;

```

Figure 3.13: Pseudo Code of LCE Recognition Algorithm



3.3 Example

In this section, we present how the proposed graph encoding algorithms can be applied to an example program shown in Figure 3.14, a prime number generator that generates prime numbers no larger than integer a . A two-bit message 01 will be embedded into the program.

```

int k(int);
int main()
{
    int a, b, sum; //v1
    printf("insert a prime number \n"); //v2
    scanf("%d",&a); //v3
    for(sum=0,b=2;b<=a;b++) { //v4
        if(k(b)) //v5
            printf("%3d",b); //v6
        sum+=b;
    }
    return 0;
}
int k(int b)
{
    int i;
    for(i=2;i<=b/2;i++)
        if(b%i==0)
            return 0;
    return 1;
}

```

Figure 3.14: Example Program

In the phase of transform, we select the blocks as vertices and the program is parsed into graph as

Figure 3.15. With the length of message, a four-vertex graph is prerequisite. We select a four-vertex graph including vertices set $V = \{v_1, v_3, v_5, v_6\}$ and edges set $E = \{(v_5, v_6)\}$. According the analysis of path, v_1 is selected as pivot vertex. In the embedding phase, LE algorithm is used to embed the message 01 by adding the edges (v_1, v_3) and (v_3, v_6) as Figure 3. The watermarked program and its related parsed graph is shown as Figure 4 and Figure 5 respectively.

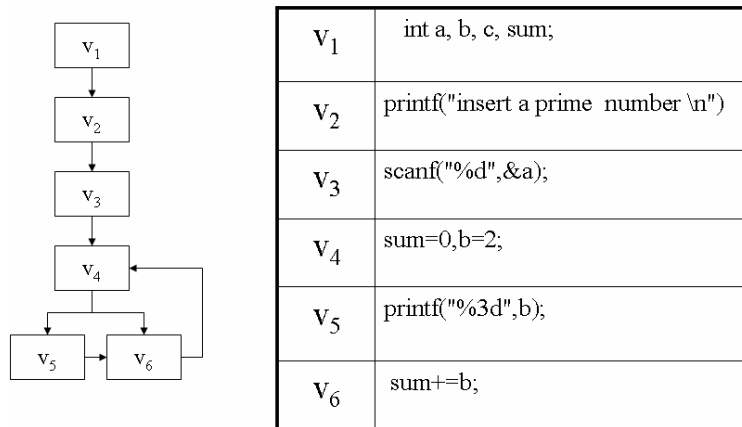


Figure 3.15: The Parsed Program

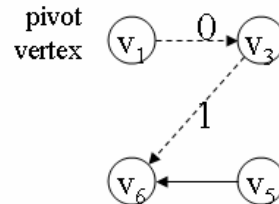


Figure 3.16: The Graph of Embedding

```

int k(int);
int main()
{
    int s, t;
    int a, b, sum; //v1
    if ((s^2+s)%2)
        goto s2
    printf("insert a prime number \n"); //v2
v3: scanf("%d",&a); //v3
    if (!(t^2+t+1)%2)
        goto v5
    for(sum=0,b=2;b<=a;b++) { //v4
        if(k(b))
v5:     printf("%3d",b); //v5
        sum+=b; //v6
    }
    return 0;
}

```

Figure 3.17: The Watermarked Program

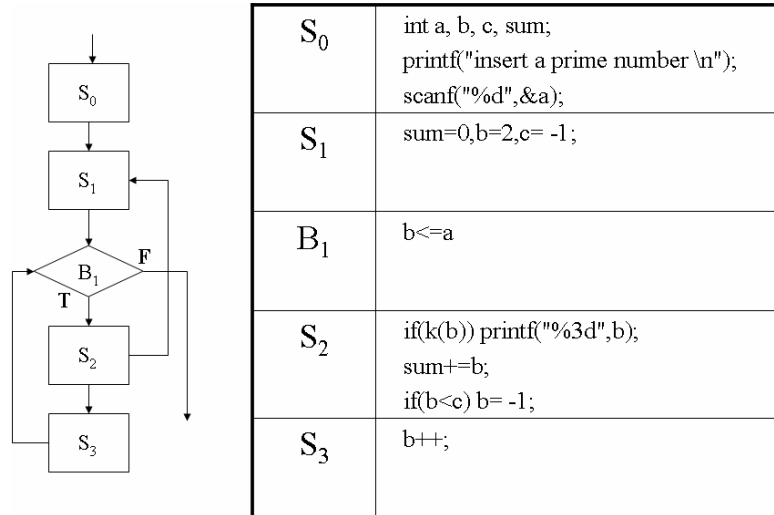


Figure 3.18: The Parsed Watermarked Program

3.4 Path Analysis

Path analysis is a useful tool which is not only finding the longer path but also providing a test and verify. Take Figure 3.19 as an example, we introduce the concept of path analysis. Original graph is given, as Figure 3.19 (a) is a 4-vertices diagram and v_1 is selected as *pivot vertex*. LE algorithm is used to embed a 2-bit message with 4-vertices diagram. For a 2-bit message, Figure 3.19 (b) shows 4 possible paths to embed 4 possible messages. Given another graph as Figure 3.20 (a), it is obvious to find that embedding process encounters problem if the message is 00 or 01. And graph as Figure 3.21 (a) can't be embedded into any message if *pivot vertex* is v_1 .

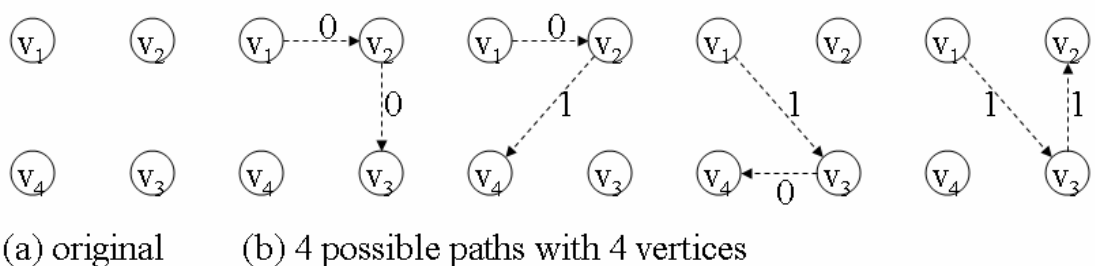


Figure 3.19 Example of 4 Possible Paths with 4 Vertices and LE Algorithm

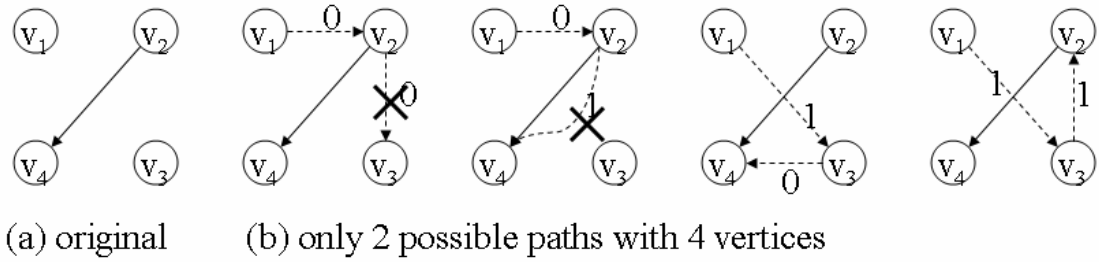


Figure 3.20 Example of 2 Possible Paths with 4 Vertices and LE Algorithm

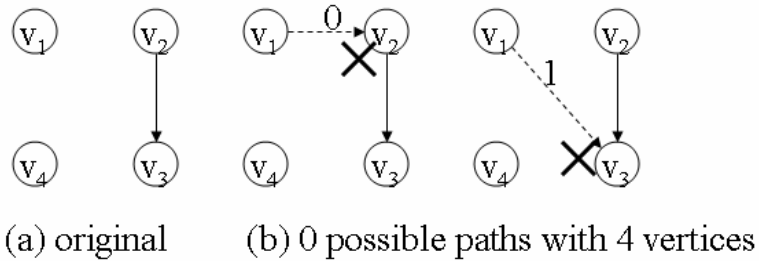


Figure 3.21 Example of Zero Possible Paths with 4 Vertices and LE Algorithm

The situation of success and failure embedding can be checked with *path analysis*. In the same way, possible paths are analyzed and found if a vertex is selected as *pivot vertex*. Take Figure 3.19 as an example, the graph diagram is as Figure 3.22 (a). The graph diagram shows the possible paths from the *pivot vertex*. From v_1 , the vertices pair (v_2, v_3) is the nearest one not connected to v_1 and (v_1, v_4) will not be a possible path. The graph diagram will expand upon a tree diagram as Fig 3.22 (b) under the rule of LE encoding algorithm. In tree diagram, it is obvious to find four possible paths which are corresponding to Figure 3.19 (b).

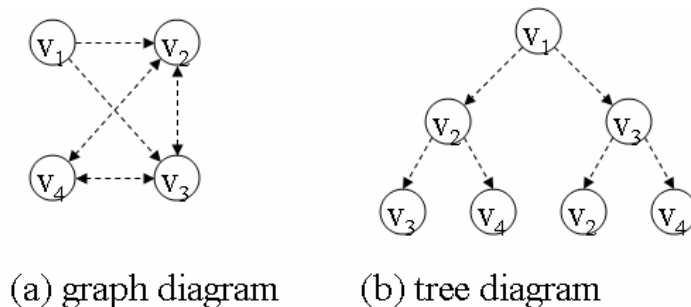


Figure 3.22 Example of Path Analysis with 4 Vertices and LE Algorithm

Figure 3.23 is another example corresponding to Figure 3.20. The graph diagram shows the possible

paths from the *pivot vertex*, v_1 . The tree diagram as Fig 3.23 (b) will expand from graph diagram. There are only two possible paths under the rule of LE encoding algorithm.

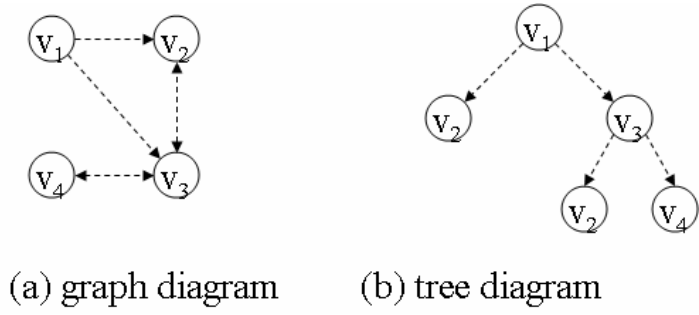


Figure 3.23 Example of Path Analysis with 2 Vertices and LE Algorithm

3.5 Error Detection

Path analysis described in the previous section can also be applied in error detection. According to the information of pivot vertex and remaining vertex, we can verify that some bits of message are in error or not. Figure 3.24 shows an example of error detection:

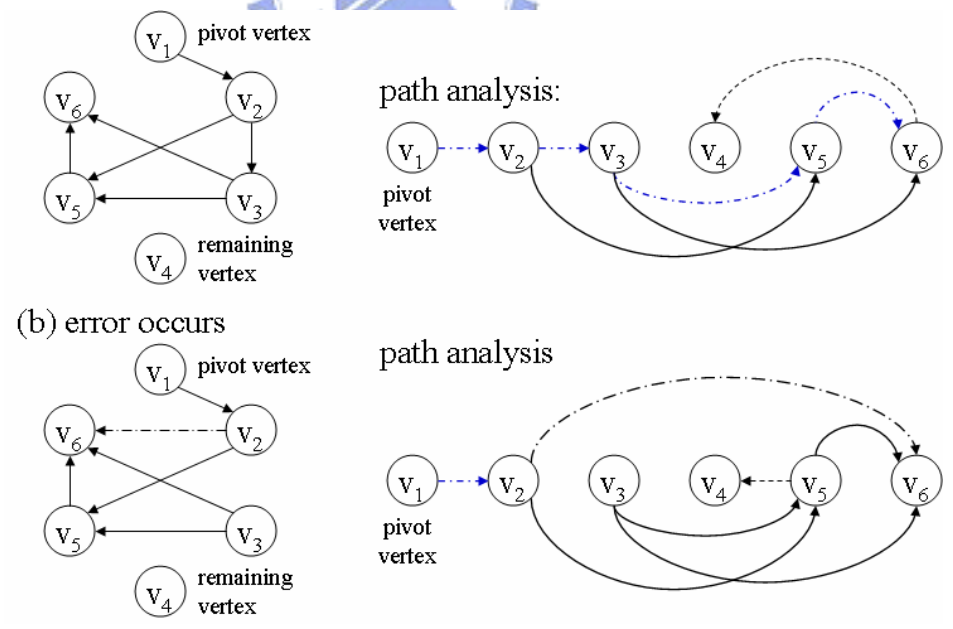
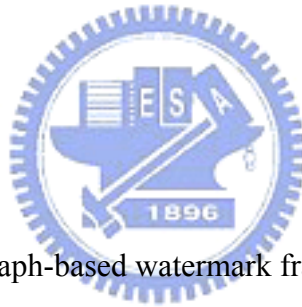


Figure 3.24 Example of Error Detection

Given watermarked graph G' with LE algorithm as Figure 3.24 (a), start and remaining vertex are v_1 and v_4 respectively. From the pivot vertex v_1 , to fit in with the information of remaining v_4 , an

embedding path $E = \{(v_1, v_2), (v_2, v_3), (v_3, v_5), (v_5, v_6)\}$ is found. A Hamiltonian path can be found if we add a virtual directed path (v_6, v_4) . The message is extracted correctly by LE recognition algorithm.

Figure 3.24 (b) shows that an error occurs in edge (v_2, v_3) and an altered edge (v_2, v_6) instead. From the pivot vertex v_1 , with the information of remaining v_4 , we can't find any possible path. Critical error occurs in graph if there is Hamiltonian path is found and we can use path analysis to recover the message. At first, from LE embedding algorithm, we analyze the graph and find that edge (v_1, v_2) should be correct message. The next pivot vertex is v_2 and $\{v_3, v_4\}$ is the nearest vertices pair and there should be an edge (v_2, v_3) or (v_2, v_4) . According to the information of the edge (v_3, v_5) , we can recover the edge (v_2, v_3) and find the correct embedding path. A Hamiltonian path is found if we add a virtual directed path (v_5, v_4) . The message are recovered and extracted correctly by LE recognition algorithm.



3.6 Summary

In this chapter, we propose a graph-based watermark framework can adopt more than three kinds of graph encoding algorithms. According to the requirement of robust and bit rate, LE, CE and LCE algorithm can be applied respectively.

Chapter 4

Analysis

After illustrating the proposed framework and algorithms, analysis and comparison with other framework and algorithms is an essential work. The performance in stealth, robust and flexibility are the criteria for software watermark algorithms.

4.1 Security Analysis

After the watermarked graph is produced, there are many kinds of adversaries. A robust watermark algorithm can prevent specific attack from extracting or destroying the embedded message. We focus on preventing the additive/subtractive attacks since the graph is constructed by vertices and edges. Thus, a graph-based watermark is vulnerable to attacks on the edges and vertices in the graph. In this section, we illustrate the software watermark attacks on graph edges and vertices.

◆ Edges additive/subtractive attack

The path analysis described in the previous section can be used to detect if there is any redundant or missing edges, which can result in destroying the watermarked information in the software module. Before the process of extraction, embedding path must be found to construct a Hamiltonian path. Single edge has been altered will be recovered as we described in last section. If few bits have been altered, we have to compare with the information of path analysis from original graph according to start and remaining vertex and graph encoding algorithm. Possible paths will be found to recover the message during comparison.

◆ Vertices additive/subtractive attack

The number of vertices is restricted by the number of bit of the message. For an example, N vertices must be matched with $N+2$ bit message for LE algorithm. Redundant or lost vertices will be detected by the character of graph encoding algorithm. To recover the message, we

have to compare with the information of path analysis from original graph according to start and remaining vertex and graph encoding algorithm. During comparison, we will find the variation of vertices and reconstruct the graph to extract the correct message.

Figure 4.1 is an example of vertex subtractive attack. Adversaries have deleted the v_6 , and the edges (v_3, v_6) and (v_5, v_6) become null pointers. It is obviously that a vertex had been deleted or lost. The correct message can still be extracted by rebuilding the graph.

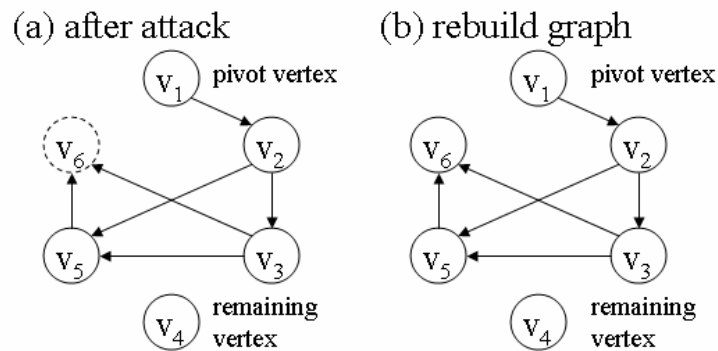


Figure 4.1 Example of Vertex Subtractive Attack

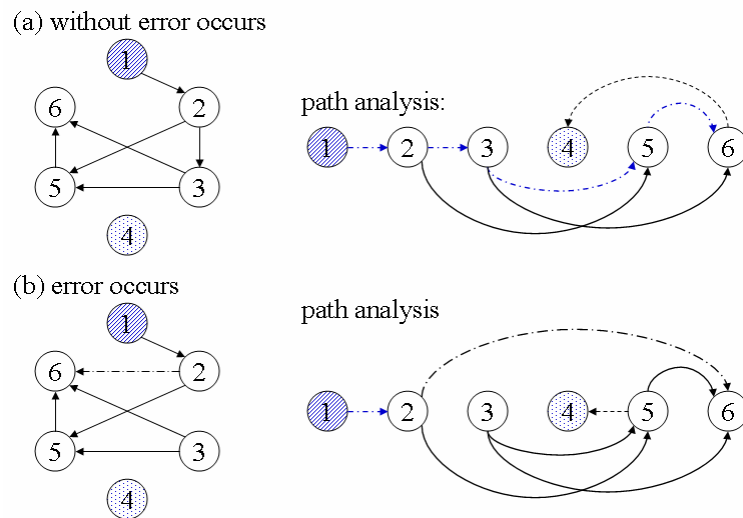


Figure 4.2 Example of Edge-flip Attack

◆ Edge-flip Attack

An edge-flip attack against the watermark reorders the edge between vertices. The outgoing of edge will be redirected to other vertices in the graph. Figure 4.2 (a) is an example of the model of attack. Adversaries try to break watermark by changing the edge (v_2, v_3) to be the edge (v_2, v_6) . The malicious attack can be detected by path analysis. According the LE algorithm, the

original graph can be recovered. The message of watermark is still extractable correctly.

◆ Vertex-Split Attack

The model of vertex-split attack splits the nodes in the graph. Each node will be divided into two vertices. New edges are connected between the divided vertices. With the information, the number of extracted message should be equal to embedded message in each graph or subgraph. The redundant bits of message will be found by path analysis. We can extract the broken message with some redundant 0 bits. However, we can't recovery the correct value efficiently without the information of original graph.

4.2 Bit rate Analysis

Bit rate is an important character to estimate the performance of stealth of watermark algorithm. The definition which is given mostly is the ratio of code size of watermark to watermarked program. However, we can find that bit rate is affected by factor such as programming language, syntax of programming language...etc. Different kinds of algorithms which are used to construct the encoded graph will make bit rate have different result in comparison with other graph encoding algorithm. We want to focus on constructing the encoded graph and neglect the difference in implement, the definition is given below: The ratio of number of bits encoded by adding edges to the total size of watermarked graph. With the definition of bit rate, we will analyze each graph encoding algorithm respectively.

Given a watermark M_B contains n bits and graph is constructed by nodes and edges. The number of byte of graph, $B = S_n \times N_n + S_e \times N_e$, where N_n and N_e are number of node and edge respectively, S_n and S_e are size of node and edge respectively. The number of byte of watermarked graph is $B' = S_n \times N_n + S_e \times N'_e$ and N'_e is the number of edge after watermark being embedded. According to definition, bit rate is defined as $(B' - B)/B'$.

Adding Edge Encoding (QP Algorithm) and Color Encoding:

It is not easy to calculate the correct value, estimation the bit rate of the common case instead. When the number of node $N_n = N$, the increasing number of edge is estimated as $N'_e - N_e = N - 1$. Based on the assumption that there will be no existed edge before embedding phase, bit rate is $(B' - B)/B' = (N - 1)S_e / (NS_n + (N - 1)S_e)$.

Link Encoding and LC Encoding:

When the number of node $N_n = N$, the increasing number of edge is $N'_e - N_e = N - 2$. Bit rate will be $(B' - B)/B' = (N - 2)S_e / (NS_n + (N - 2)S_e)$ based on the assumption that there is no existed edge before embedding phase.

Permutation Encoding and Radix Encoding:

When the number of node contains the node of root is $N_n = N + 1$, the number of edge is $N_e = 2N + 1$ and the increased number of edge is $N'_e - N_e = N$. Bit rate is $(B' - B)/B' = NS_e / ((N + 1)S_n + (2N + 1)S_e)$

Parent-pointer Tree Encoding:

It is also not easy to calculate the correct value of bit rate. When the number of node $N_n = N$, the increasing number of edge is estimated as $N'_e - N_e = N - 1$ and bit rate is $(B' - B)/B' = (N - 1)S_e / (NS_n + (N - 1)S_e)$.



4.3 Summary

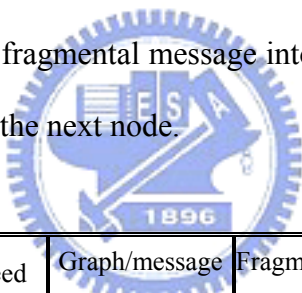
We analyze the characteristics of our graph encoding algorithms. The performance in error detection and robust is better than QP and QPS algorithm. The bit rate of each graph encoding algorithm is described in section 4.3. We will give a further discussion in next chapter.

Chapter 5

Comparison

5.1 Framework Characteristic

We adapt and modify the *Graph Theoretic Approach* which is illustrated in section 2.1. The differences between these two frameworks are the secret key and random walk in merging step. There are three secret key for partition the graph in clustering step, choose the next node in merging step and encrypt the extracted value in recovery step respectively. In our proposed framework, we use only one secret key to segment the message into fragmental message and hence the segment of subgraph will be decided. The second difference is the random walk for merging watermark and graph. We embed each ordered fragmental message into each ordered subgraph respectively instead of using random seed to decide the next node.



	Random seed	Graph/message segment	Fragmental message generating	vertices select	well connected graph
<i>Graph Theoretic Approach</i>	3	yes	random	random	yes
Proposed Framework	1	yes	random / user	rule	yes

Table 1 : Framework Characteristic

5.2 Bit rate of Software Watermark Algorithms

As we discuss in section 4.3, bit rate of each graph encoding is analyzed, and furthermore, we will give an example for illustrating the comparison. Given an integer as watermark $W = 10000_{10} = 10011100010000_2$ and $|W| = 14$. Let $S_n = kS_e$, bit rate is show as following:

QP Algorithm/QPS Algorithm:

$(B' - B)/B' = (N - 1)S_e / (NS_n + (N - 1)S_e) = (N - 1) / ((k + 1)N - 1)$. Let $N=15$ from the bit number is 14 after estimation, and $(B' - B)/B' = 14 / (15k + 14)$.

Link Encoding:

$(B' - B)/B' = (N - 2)S_e / (NS_n + (N - 2)S_e) = (N - 2) / (N(k + 1) - 2)$. If the bit number is 14, $N=16$ is found, and $(B' - B)/B' = 14 / (16k + 14)$.

Color Encoding:

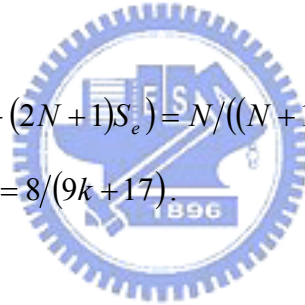
$(B' - B)/B' = (N - 1)S_e / (NS_n + (N - 1)S_e) = (N - 1) / ((k + 1)N - 1)$. Let $N=8$ from the bit number is 14 after estimation, and $(B' - B)/B' = 7 / (8k + 7)$

Link with Encoding:

$(B' - B)/B' = (N - 2)S_e / (NS_n + (N - 2)S_e) = (N - 2) / (N(k + 1) - 2)$. If the bit number is 14, $N=9$ is found, and $(B' - B)/B' = 7 / (9k + 7)$.

Permutation Encoding:

$(B' - B)/B' = NS_e / ((N + 1)S_n + (2N + 1)S_e) = N / ((N + 1)k + (2N + 1))$. $N = 8$ is found if the bit number is 14, and $(B' - B)/B' = 8 / (9k + 17)$.



Radix Encoding:

$(B' - B)/B' = NS_e / ((N + 1)S_n + (2N + 1)S_e) = N / ((N + 1)k + (2N + 1))$. $N = 7$ is found if the bit number is 14, and $(B' - B)/B' = 7 / (8k + 15)$.

Parent-pointer Tree Encoding:

$(B' - B)/B' = (N - 1)S_e / (NS_n + (N - 1)S_e) = (N - 1) / (Nk + (N - 1))$ Let $N=8$ from the bit number is 14 after estimation, and $(B' - B)/B' = 15 / (16k + 15)$.

According the definition, we know that ability of resilience is in inverse proportion to bit rate. In Table 2, by applying different k , we will find that the difference between each algorithm in bit rate isn't affected. In the same situation, Radix encoding and Permutation encoding algorithms have the better performance in bit rate than LE and LCE algorithm, and QP/QPS and PP tree algorithms have worse performance in bit rate. From the information in table, we can find that "list-based"

algorithms have better performance than “graph-based” algorithms.

	$k = 2$	$k = 3$	$k = 4$
Radix	0.225	0.180	0.149
Permutation	0.228	0.182	0.151
PP trees	0.319	0.238	0.190
QP / QPS	0.318	0.238	0.189
Link Encoding	0.304	0.225	0.179
Color Encoding	0.304	0.225	0.179
LCE	0.280	0.205	0.163

Table 2: Bit rate of Graph Encoding Algorithm

5.3 Characteristic of Software Watermark Algorithms

	<i>stealth</i>	<i>Flexible path selecting</i>	<i>Error Detection</i>
<i>Radix</i>	<i>Good</i>	<i>No</i>	<i>No</i>
<i>Permutation</i>	<i>Good</i>	<i>No</i>	<i>Yes</i>
<i>PP trees</i>	<i>Bad</i>	<i>No</i>	<i>No</i>
<i>QP</i>	<i>Bad</i>	<i>No</i>	<i>Yes</i>
<i>QPS</i>	<i>Bad</i>	<i>No</i>	<i>No</i>
<i>Link Encoding</i>	<i>Fair</i>	<i>Yes</i>	<i>Yes</i>
<i>Color Encoding</i>	<i>Fair</i>	<i>Yes</i>	<i>No</i>
<i>LCE</i>	<i>Fair</i>	<i>Yes</i>	<i>Yes</i>

Table 3: Comparison of Graph Encoding Algorithm

We compare each graph encoding algorithm with bit rate, flexible path selecting and the ability of single error detection. It is obvious that Radix have higher bit rate but with worst performance in robust and flexibility. LCE and LE will be ideal algorithm if we want have requirement of stealth, flexibility and robust.

5.4 Summary

In this chapter, we analyze the difference between proposed and related framework. The detail characteristic of each algorithm is also describes in Table 5.3.



Chapter 6

Conclusion

In this paper, we modify and build a flexible software watermarking framework which can adopt graph theoretical encoding algorithms. In this framework, watermark and program will be parsed into graph and merged with graph-based encoding algorithms. The hidden message with graph encodings is harder to be detected and attacked by adversaries. Program with different graph encoded watermark will have different performance in resilience and robust. QP and QPS algorithms have neither credibility nor resilience and robust. Although CT algorithm have high robust, embedding and recognition in execution time is hard and time-consuming to be constructed with non-Java language. Based on these algorithms, we design and modify three kinds of algorithms can improve resilience and robust respectively by link-type structure and application of color. Besides, proposed path analysis is an important analysis for selecting the longest embedding path and searching and analyzing the correct path. With path analysis, the watermark can be easier to be constructed and extracted, and the information of start and rest vertices can output instead of the whole information of original graph. Decrement of storage will improve the convenience and practicability for mobile code system.

Chapter 7

Future Work

We believe that proposed framework and graph encoding algorithms have improvement in stealth and robust. The framework can adopt not only our proposed but also other kinds of graph encoding algorithms, such as QP and QPS. Three kinds of graph encoding algorithms still have another kind of method to modify the integer of watermark into a shorter message in bit stream. If the message is shorter, the performance of stealth and resilience can be better. Encoding algorithm will be improved with robust is carried into effect. Implement is also an important work that we should achieve for verify the characteristics of each algorithm and make more improvements.



Reference

- [1] Robert L. Davidson. and Nathan Myhrvold. “Method and system for generating and auditing a signature for a computer program,” *US Patent 5,559,884*, September 1996. Assignee: Microsoft Corporation.
- [2] Gang Qu and Miodrag Potkonjak. “Hiding Signatures in Graph Coloring Solutions,” In *CA90095*. USA. 1999
- [3] Ginger Myles. and Christian Collberg. “Software Watermarking Through Register Allocation: Implementation, Analysis, and Attack,” *ICISC 2003*, LNCS 2971, pp. 274–293, 2004.
- [4] C. Collberg and C. Thomborson. “Software watermarking: Models and dynamic embeddings,” In *POPL '99*, Jan. 1999.
- [5] Ramarathnam Venkatesan. Vijay Vazirani. Saurabh Sinha. “A Graph Theoretic Approach to Software Watermarking,” In *4th International Information Hiding Workshop*, Pittsbutgh, PA, April 2001.
- [6] Lin Yuan, Gang Qu, Ankur Srivastava. “VLSI CAD tool protection by birthmarking design solutions,” In *Great Lakes Symposium on VLSI, Proceedings of the 15th ACM Great Lakes symposium on VLSI table of contents*, Chicago, Illinois, USA, Pages: 341 - 344, 2005
- [7] Christian Collberg, Edward Carter, Stephen Kobourov, and Clark Thomborson. Error-correcting graphs. *Workshop on Graphs in Computer Science (WG'2003)*, June 2003.
- [8] Christian Collberg, Clark Thomborson, and Douglas Low. Manufacturing cheap, resilient, and stralthy opaque construct. In *Principles of Programming Languages 1998*,
- [9] Fabien A.P. Peticolas, Ross J. Anderson, and Markus G. Kuhn. Attacks on copyright marking systems. In *Second Workshop On Information Hinding*, Portland, Oregon, April 1998.
- [10] Tapas Sahoo and Christian Collberg. Software Watermark in the frequency domain: Implementation. Analysis, and attacks. Technical Report TR04-07, Department of Computer Science, University of Arizona, March 2004.

- [11] Ginger Myles, Christian Collberg. "K-gram based software birthmarks," In *Symposium on Applied Computing, Proceedings of the 2005 ACM symposium on Applied computing*, Santa Fe, New Mexico, Pages 314 - 318, 2005
- [12] H. Berghel and L. O'Gorman. "Protecting ownership rights through digital watermarking," *IRRR computer*, 29(7):101-103, 1996.
- [13] Christian Collberg, Clark Thomborson, and Douglas Low. "A taxonomy of obfuscating transformations," *Technical Report 148, Department of Computer Science, UNiversity of Auckland*, July 1997.
- [14] Takeshi Kakimoto, Akito Monden, Yasutaka Kamei, Haruaki Tamada, Masateru Tsunoda, Ken-ichi Matsumoto. "MSR-challenge report: Using software birthmarks to identify similar classes and major functionalities ," Proceedings of the 2006 international workshop on Mining software repositories MSR '06
- [15] David Nagy-Fraksa. The easter egg archive. <http://www.eeggs.com/lr.html>, 1998.
- [16] D. Aucsmith, "Tamper Resistant Software: An Implementation," *Information Hiding, First Int'l Workshop*, R.J. Anderson, ed., pp.317-333, May 1996.
- [17] F. Hohl, "A Framework to Protect Mobile Agents by Using Reference Status," Proc. 20th Int'l Conf. Distributed Computing Systems, pp. 410-417, 2000.
- [18] Yan Zhu, Wei Zou, and Xinshan Zhu. "Collusion Secure Convolutional Fingerprinting Information Codes," In *ASIACCS' 06*, Taipei, Taiwan, March 21-24, 2006.
- [19] Sam Michiels, Kristof Verslype, Wouter Joosen, Bart De Decker. "Software issues: Towards a software architecture for DRM," In *Proceedings of the 5th ACM workshop on Digital rights management DRM '05..*
- [20] Gaurav Gupta, Jasef Pieprzyk, Huaxiong Wang. "An Attack-Localizing Watermarking Scheme for Natural Language Document," In *ASIACCS' 06*, Taipei, Taiwan, March 21-24, 2006.
- [21] Soo-Chang Pei, Yi-Chong Zeng. "Tamper Proofing and Attack Identification of Corrupted Image by using Semi-fragile Multiple-watermarking Algorithm," In *ASIACCS' 06*, Taipei, Taiwan, March 21-24, 2006.

[22] Gross Yellen. “Graph Theory and its Application,” 1999.

[23] Ingemar J. Cox, Matthew L. Miller, Jeffrey A. Bloom. “Digital Watermarking,” 2002.

[24] Lowell W. Beineke, Robin J. Wilson. “Graph Connections,” 1997

